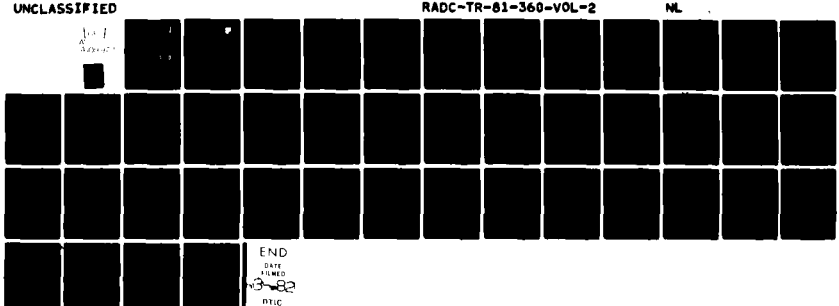
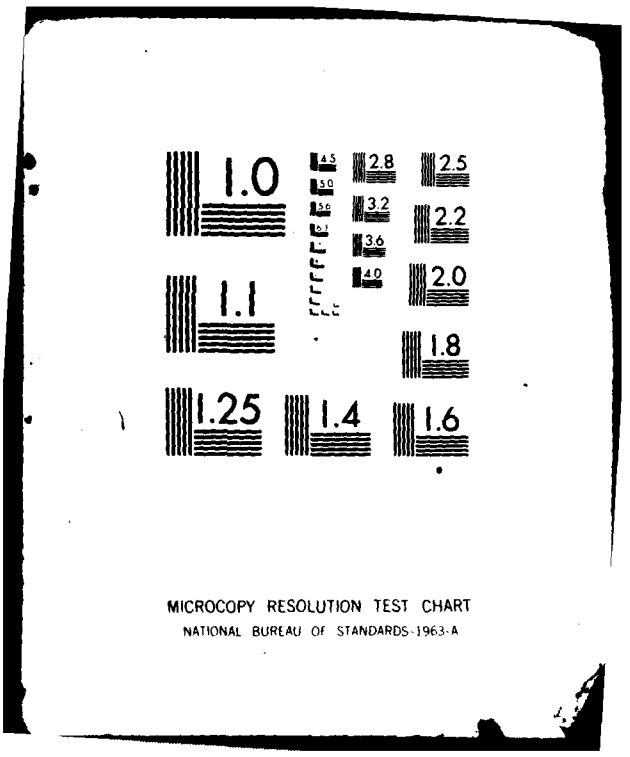
AD-A109 977 TEXAS INSTRUMENTS INC LEWISVILLE EQUIPMENT GROUP F/G 9/2
ADA INTEGRATED ENVIRONMENT III COMPUTER PROGRAM DEVELOPMENT SPE--ETC(U)
DEC 81 F30602-80-C-0293
UNCLASSIFIED RADC-TR-81-360-VOL-2 NL



END
DATE FILMED
3-82
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

PHOTOGRAPH THIS SHEET

AD A109977

DTIC ACCESSION NUMBER

LEVEL

INVENTORY

Texas Instruments, Inc
Lewisville, TX Equipment Group-ACSL

ADA Integrated Environment III Computer Program Development Specification, Interim Rept. 15 Sep. '80-15 March '81

DOCUMENT IDENTIFICATION

RADC-TR-81-360, Vol. II Dec. 81

Contract F30602-80-C-0293

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR	
NTIS	GRA&I
DTIC	TAB
UNANNOUNCED	
JUSTIFICATION	
BY	
DISTRIBUTION /	
AVAILABILITY CODES	
DIST	AVAIL AND/OR SPECIAL
A	

DTIC COPY INSPECTED 3

DISTRIBUTION STAMP

DTIC
ELECTE
S JAN 25 1982 D
D

DATE ACCESSIONED

82 01 12 009

DATE RECEIVED IN DTIC

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDA-2

DOCUMENT PROCESSING SHEET

DTIC FORM OCT 79 70A

AD A109977

RADC-TR-81-360, Vol II (of four)
Interim Report
December 1981



ADA INTEGRATED ENVIRONMENT III COMPUTER PROGRAM DEVELOPMENT SPECIFICATION

Texas Instruments, Inc.

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-360, Volume II (of four) has been reviewed and is approved for publication.

APPROVED: Elizabeth S. Kean

ELIZABETH S. KEAN
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-360, Vol II (of four)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ADA INTEGRATED ENVIRONMENT III COMPUTER PROGRAM DEVELOPMENT SPECIFICATION		5. TYPE OF REPORT & PERIOD COVERED Interim Report 15 Sep 80 - 15 Mar 81
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s)		8. CONTRACT OR GRANT NUMBER(s) F30602-80-C-0293
9. PERFORMING ORGANIZATION NAME AND ADDRESS Texas Instruments Incorporated Equipment Group-ACSL, P O Box 405, M.S. 3407 Lewisville TX 75067		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62204F/33126F/62702F 55811919
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COES) Griffiss AFB NY 13441		12. REPORT DATE December 1981
		13. NUMBER OF PAGES 34
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Elizabeth S. Kean (COES)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ada	MAPSE	AIE
Compiler	Kernel	Integrated environment
Database	Debugger	Editor
KAPSE	APSE	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Ada Integrated Environment (AIE) consists of a set of software tools intended to support design, development and maintenance of embedded computer software. A significant portion of an AIE includes software systems and tools residing and executing on a host computer (or set of computers). This set is known as an Ada Programming Support Environment (APSE). This B-5 Specification describes, in detail, the design for a minimal APSE, called a MAPSE. The MAPSE is the foundation upon which an

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 68 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

APSE is built and will provide comprehensive support throughout the design, development and maintenance of Ada software. The MAPSE tools described in this specification include an Ada compiler, linker/loader, debugger, editor, and configuration management tools. The kernel (KAPSE) will provide the interfaces (user, host, tool), database support, and facilities for executing Ada programs (runtime support system).

TABLE of CONTENTS

Paragraph	Title	Page
	SECTION 1 SCOPE	
1.1	Identification	1-1
1.2	Functional Summary	1-1
	SECTION 2 APPLICABLE DOCUMENTS	
2.1	Program Definition Documents	2-1
2.2	Military Specifications and Standards	2-1
	SECTION 3 REQUIREMENTS	
3.1	Introduction	3-1
3.1.1	General Description	3-1
3.1.1.1	Database Objects	3-1
3.1.1.2	Database Directory and Dictionary	3-2
3.1.1.3	Attributes of Database Objects	3-2
3.1.1.4	Definition and Maintenance of Attributes	3-3
3.1.1.4.1	History Attribute	3-4
3.1.1.4.2	Category Attribute	3-4
3.1.1.4.3	Access Attribute	3-4
3.1.1.5	Project and Configuration Management	3-5
3.1.1.5.1	Version Control	3-5
3.1.1.5.2	Configuration Definition	3-5
3.1.1.5.3	Ada Program Libraries	3-6
3.1.1.5.4	Partitions	3-6
3.1.1.5.5	Database Archiving	3-6
3.1.2	Interface Identification	3-6
3.1.3	Computer Interface Block Diagram	3-7
3.1.4	Program Interfaces	3-10
3.1.4.1	Interface with Programs	3-10
3.1.4.2	Interface with I/O Component of KAPSE	3-11
3.1.4.3	Interface with Command Language Interpreter	3-11
3.1.5	Function Descriptions	3-12
3.2	Detailed Functional Requirements	3-13
3.2.1	Database Interface - Introduction	3-13

3.2.1.1 Function ATTRIBUTE 3-14
3.2.1.2 Function RELATIONS 3-14
3.2.1.3 Procedure SET_ATTRIBUTE 3-14
3.2.1.4 Procedure SELECT_OBJECTS 3-15
3.2.1.5 Procedure MAKE_RELATION 3-15
3.2.1.6 Procedure BREAK_RELATION 3-15
3.2.2 Database Directory Manager - Introduction 3-15
3.2.2.1 Procedure FIND_ENTRY 3-16
3.2.2.2 Procedure ADD_ENTRY 3-16
3.2.2.3 Procedure DELETE_ENTRY 3-16
3.2.2.4 Procedure UPDATE_HISTORY 3-16
3.2.3 Database Dictionary Utility - Introduction . . . 3-17
3.2.3.1 Procedure DEFINE_ATTRIBUTE 3-17
3.2.3.2 Procedure CHECK_ATTRIBUTE 3-18
3.2.4 Database Archive Manager - Introduction 3-18
3.2.4.1 Package ARCHIVE_MANAGER 3-18
3.2.4.1.1 Structure of the Archive File 3-20
3.2.4.1.2 Procedure INITIALIZE 3-22
3.2.4.1.3 Procedure UPDATE 3-22
3.2.4.1.4 Procedure EXTRACT 3-22
3.2.5 Database Utilities - Introduction 3-22
3.2.5.1 Procedure SHOW 3-22
3.2.5.2 Procedure COPY 3-23
3.2.5.3 Procedure COMPARE 3-23
3.2.5.4 Procedure BACKUP 3-23
3.2.5.5 Procedure RESTORE 3-23
3.2.5.6 Procedure VERIFY 3-24
3.2.6 User Information Manager - Introduction 3-24
3.2.6.1 Function USER_ATTRIBUTE 3-25
3.2.6.2 Function USER_RELATIONS 3-25
3.2.6.3 Procedure CHECK_USER_ATTRIBUTE 3-25
3.2.6.4 Procedure FIND_USER_ENTRY 3-26
3.2.7 User Information Utility - Introduction 3-26
3.2.7.1 Procedure ADD_USER 3-26
3.2.7.2 Procedure DELETE_USER 3-26
3.2.7.3 Procedure DEFINE_USER_ATTRIBUTE 3-27
3.2.7.4 Procedure SET_USER_ATTRIBUTE 3-27
3.2.7.5 Procedure MAKE_USER_RELATION 3-27
3.2.7.6 Procedure BREAK_USER_RELATION 3-28
3.2.8 Host Filesystem Interfaces - Introduction 3-28
3.2.8.1 High-Level I/O Interface 3-28
3.2.8.2 Low-Level I/O Interface 3-28
3.2.8.3 System Tables 3-29

SECTION 4 QUALITY ASSURANCE PROVISIONS

4.1 Introduction 4-1
4.1.1 Computer Program Component Test and Evaluation . 4-1

4.1.2 Integration Testing 4-2
4.1.3 Formal Acceptance Testing 4-2
4.2 Test Requirements 4-2
4.2.1 Function Tests 4-2
4.2.2 Utility Tests 4-2
4.2.3 Rehosting Tests 4-3
4.3 Acceptance Test Requirements 4-3
4.3.1 Performance Requirements 4-3
4.4 Independent Validation and Verification 4-3

APPENDIX A REFERENCES

LIST of FIGURES

Figure	Title	Page
3-1	Database Directory Interfaces.	3-8
3-2	User Information Interfaces.	3-9
3-3	Structure of File Attribute Block Fixed Area	3-11
3-4	Archive File Process	3-19
3-5	Structure of the Archive File.	3-21

SECTION 1

SCOPE

1.1 Identification

This part of this specification establishes the requirements for performance, design, test, and acceptance of a computer program configuration item identified as the Ada Database Subsystem, CPCI C02. This CPCI contains the following computer program components:

* C02.01 Database Interface (DI)
* C02.02 Database Directory Manager (DDM)
* C02.03 Database Dictionary Utility (DDU)
* C02.04 Database Archive Manager (DAM)
* C02.05 Database Utilities (DUs)
* C02.06 User Information Manager (UIM)
* C02.07 User Information Utility (UIU)
* C02.08 Host Filesystem Interfaces (HFI)

The Database Subsystem is provided on all computer systems supporting the Ada Integrated Environment.

1.2 Functional Summary

The purposes of this specification are:

1. To specify the functions of the Database Subsystem.
2. To describe the interfaces between the Database Subsystem and other components of the Ada Integrated Environment.

The function of the Database Subsystem is to manage all user and system-generated data in the form of database objects, and to manage higher-level information concerning each of these objects in the form of database directories and dictionaries. The Database Subsystem also manages information concerning users of the system and uses this information to control access to database objects and APSE tools.

SECTION 2

APPLICABLE DOCUMENTS

2.1 Program Definition Documents

[DoD80A] Requirements for Ada Programming Support Environments: "STONEMAN", DoD (February 1980).

[RADC80] Revised Statement of Work for Ada Integrated Environments, RADC, Griffiss Air Force Base, NY (March 1980).

[SOFT80A] Ada Compiler Validation Capability: Long Range Plan, SofTech Inc., Waltham, MA (February 1980).

[SOFT80B] Draft Ada Compiler Validation Implementers' Guide, SofTech Inc., Waltham, MA (October 1980).

2.2 Military Specifications and Standards

[DoD80B] Reference Manual for the Ada Programming Language: Proposed Standard Document, DoD (July 1980) (reprinted November 1980).

SECTION 3

REQUIREMENTS

3.1 Introduction

The Ada Database Subsystem (ADS) is the central feature of the Ada Integrated Environment. It serves as the repository for all user- and system-generated data and as a primary medium of communication between APSE components.

3.1.1 General Description

The Ada Database Subsystem provides a flexible, efficient, automated and programmable storage mechanism as the foundation of a comprehensive software support facility. It is sufficiently general to provide capabilities for the storage and retrieval of information concerning all components of a software product.

3.1.1.1 Database Objects

Each separately identifiable collection of information in the database is called an object. Each object has a unique name; it has attributes that describe its nature and its relationships with other objects, and it contains information. APSE tools and users have controlled access to database objects and controlled capabilities to create, delete, modify, store, archive, and retrieve them.

Each database object is uniquely identified by two distinct names. The pathname of an object is the user assigned sequence of Ada identifiers that specifies the path through the directory hierarchy from the base or root to the specific database object. The database name (DB_name) of an object is the unique internal name of an object assigned by the system when the object is created; this name is never changed. The one-to-one mapping of DB_names to pathnames is found in a system-maintained table called the Database Map.

A complete database object consists of its content, its attributes, and a directory entry.

* The content of an object, or the raw information it contains, is determined entirely by the APSE tool or user program that creates the object. This data is stored as an unstructured file in space allocated by the host file system.

* The attributes of an object are determined partly by the APSE Executive and software tools and partly by APSE users or managers.

This data is stored separately from the content of the object, in specially structured records, and may be directly accessed only by the Database Directory Manager.

* The directory entry associated with an object is a specially structured record in a file (also an object) accessible only to the Directory Manager. The principal function of a directory entry is to provide the mapping from the name of a database object to the location of its contents and its attributes.

Any object, including a directory, may be separated into physically noncontiguous areas for efficiency of storage management. The mapping of logical records to physical addresses is handled entirely by the database system and host filesystem and is transparent to APSE tools and users.

3.1.1.2 Database Directory and Dictionary

All database objects and attribute records are located through a directory which interfaces with APSE tools and users through the Directory Manager. This single port to the database allows centralized verification of access rights and facilitates the collection of historical and statistical information for all database objects. The directory is a hierarchically organized structure of unlimited depth; this provides the flexible organization desired in a software development environment.

The database dictionary defines the properties of the attributes of database objects within the domain of the directory. A dictionary entry for an attribute includes its name, data type, access restrictions and other appropriate information corresponding to that found in a symbol table. Certain system-defined attributes will be defined by dictionary entries at system construction. Users may also define attributes, and utilities are provided for this purpose. User access to the dictionary is controlled by the Database Dictionary Utility.

3.1.1.3 Attributes of Database Objects

An attribute of a database object is a descriptor of the nature of an object. Each attribute has a name and value. The value of an attribute may be one of the following data types:

* INTEGER A signed numeric value
* BOOLEAN A True or False value
* STRING A character string
* ENUMERATED One of a predefined set of values
* RELATION A list of pointers to other database objects

Every object has attributes. However, the set of applicable attributes may vary from object to object.

Conceptually, a relation is a labeled, directed arc that connects any two database objects. Each relation has a unique name. Any database object may be connected to one or more others by a single relation, thus the value of a relation may be visualized as a list of names. All relations are bidirectional; inverse relations are maintained, and relations may be followed in either forward or inverse directions.

3.1.1.4 Definition and Maintenance of Attributes

The four categories of attribute definition and maintenance are:

* System-defined, system-maintained
* System-defined, user-maintained
* User-defined, system-maintained
* User-defined, user-maintained

Certain system-defined attributes contain fundamental information to describe the physical organization and characteristics of a database object. This information is essential to the proper operation and integrity of the system. System-defined attributes include physical location, space allocation, storage organization, and access constraints. These attributes are predefined for all database objects, and their values are assigned and updated automatically by the Ada Database Subsystem. The values of these attributes are made available to authorized users and APSE tools on a read-only basis.

Another group of system-defined attributes, while not critical to system integrity, is useful for basic data mangement functions and can be updated automatically by the ADS. This group includes dates and times of creation, update, or access; record sizes and blocking factors; and access methods.

A final set of attributes may be predefined as a convenience to users of the system. Default values may be assigned at the time an object is created, and authorized users and tools may later interrogate or modify the values of these attributes as needed.

Finally, users may define and maintain attributes that are not otherwise system-maintained.

The Ada Database Subsystem includes these predefined attributes:

* History Attribute
* Category Attribute
* Access Attribute

3.1.1.4.1 History Attribute

The historical data associated with objects in the database is preserved as the history attribute. The history attribute will consist of specially structured history records whose contents are equivalent to an audit trail of modifications for each object. A history record will be created by the database subsystem whenever a database object is created or updated. Changes to the values of designated attributes are also recorded in the history records.

The information in a history record includes:

* Name of program creating or updating
* User and task identification
* Names of other input database objects involved
* Version/revision codes
* Date/time stamps

The procedure UPDATE_HISTORY is a routine in the Database Directory Manager that will:

* determine the name of programs
* determine names of all input files
* determine USER_ID
* insert version/revision codes
* construct/insert date and time stamps

3.1.1.4.2 Category Attribute

The category attribute indicates the type of information contained in an object. This implies use or access restrictions to prevent the use of certain database objects by inappropriate tools or users. The category attribute is an enumerated attribute type requiring selection of one of a predetermined set of values. Database initialization includes the initial set of category values required for proper operation of the APSE. Tools are provided to permit the system manager to extend the set of predefined values for the category or any other enumerated attribute, so that user defined values may be added.

3.1.1.4.3 Access Attribute

The APSE system includes a User Information Directory and a User Information Dictionary which contain information (user attributes) about each authorized user of the system. User attributes include access authorizations and restrictions for APSE tools, database objects, and system resources.

Access rights to each database object are indicated by an access attribute associated with the object. General access rights are indicated by an enumerated type attribute with values of:

* PUBLIC (unlimited access by others)

* LIMITED (access only by certain users)

* PRIVATE (access only by the owner of the object)

Limited access further requires a list of authorized user identifications. For each user named in this list, access rights are further specified (e.g., "read", "modify", "execute"). The Directory Manager will verify access rights before permitting any access to a database object and will raise an exception if an access violation is attempted. A privileged user has unrestricted access to all database objects.

Each database object has system-defined attributes which identify its creator and current owner. At the time of the creation of the object, the Directory Manager records the creator. The creator, (or any privileged system user) may assign a different owner.

Access to database objects may also be controlled by reference to version or other attributes in the command language programs that test such attributes before passing database objects as parameters to APSE tools or user programs.

3.1.1.5 Project and Configuration Management

The Ada Database Subsystem shall provide facilities for version control, configuration definition, program libraries, partitioning, and archiving of database objects.

3.1.1.5.1 Version Control

The Ada Database Subsystem allows the designation of a specific version, a default version, a list of versions, or all versions of a database object. This mechanism is based on the naming conventions associated with subtrees in the hierarchically structured directory and also uses the more general facility of a combination of attributes and relations within the database to establish a version group. System-defined version attributes are provided for each database object. Every object possesses date/time stamp attributes, automatically maintained by the Database Subsystem, to record "creation", "last access", and "last update" events. Assignment of values to other version attributes will be performed by authorized APSE tools and users.

3.1.1.5.2 Configuration Definition

A configuration is a collection of database objects that are related by some common property or requirement, such as the set of objects required by the binder to form an executable Ada program. Configurations can be described by ordered lists of names. Configuration descriptions use the standard

attributes, relations, and associated routines provided in the Ada Database Subsystem.

3.1.1.5.3 Ada Program Libraries

The logical grouping of compilation units making up a complete Ada program or set of related Ada programs is a special configuration called a program library. The LIBRARY_FILE_UTILITY defined in the Ada Language Processor segment allows for library definition and maintenance.

3.1.1.5.4 Partitions

The design of the database and associated access facilities provides for partitioning of the database in the following ways:

* The hierarchical structure of the database allows partitioning by subtrees, using naming conventions for the objects as elements of the partition (nodes and leaves of the subtree).

* User-defined relations provide a general graph structure, with the ability to relate any database object to an arbitrary number of other objects, or to connect a sequence of objects in an ordered relationship.

* All database objects possessing a particular attribute-value combination may be considered members of a partition.

3.1.1.5.5 Database Archiving

The Ada Database Subsystem provides the capability of archiving database objects. The Database Archive Manager (see Section 3.2.4) will retain the integrity, consistency, and availability of all database objects.

3.1.2 Interface Identification

The design of the Ada Database Subsystem is based on the requirement for the Ada Integrated Environment to be portable. The Ada Database Subsystem is designed to take advantage of those facilities provided by the host computer systems. Moving to a new system is usually costly and error-prone (HAL80). This problem is reduced through the use of a virtual operating system that disentangles the computing environment from the underlying host operating systems or primitives.

To the Ada Database Subsystem, all APSE tools as well as user programs are "Ada Programs". There is one interface package called DATABASE_INTERFACE which acts as the interface between the Database Subsystem and all Ada programs. However, the Command Language Interpreter and Low-Level I/O deserve special mention. The database interface with the human user is through the Command Language Interpreter, while the database interface with the host system is through the Low-Level I/O.

3.1.3 Computer Interface Block Diagram

Figure 3-1 shows the interfaces between the Ada Database Subsystem, Ada programs, Command Language Interpreter, Data Files and Host System I/O Devices. Figure 3-2 represents the interfaces between the Ada Database Subsystem, Command Language Interpreter, and user programs for user information queries. Certain standard data structures are used to communicate information, in the form of attributes and relations, between programs. These data structures include File Parameter Blocks (FPB), File Attribute Blocks (FAB), File Control Blocks (FCB), and User Attribute Blocks (UAB).

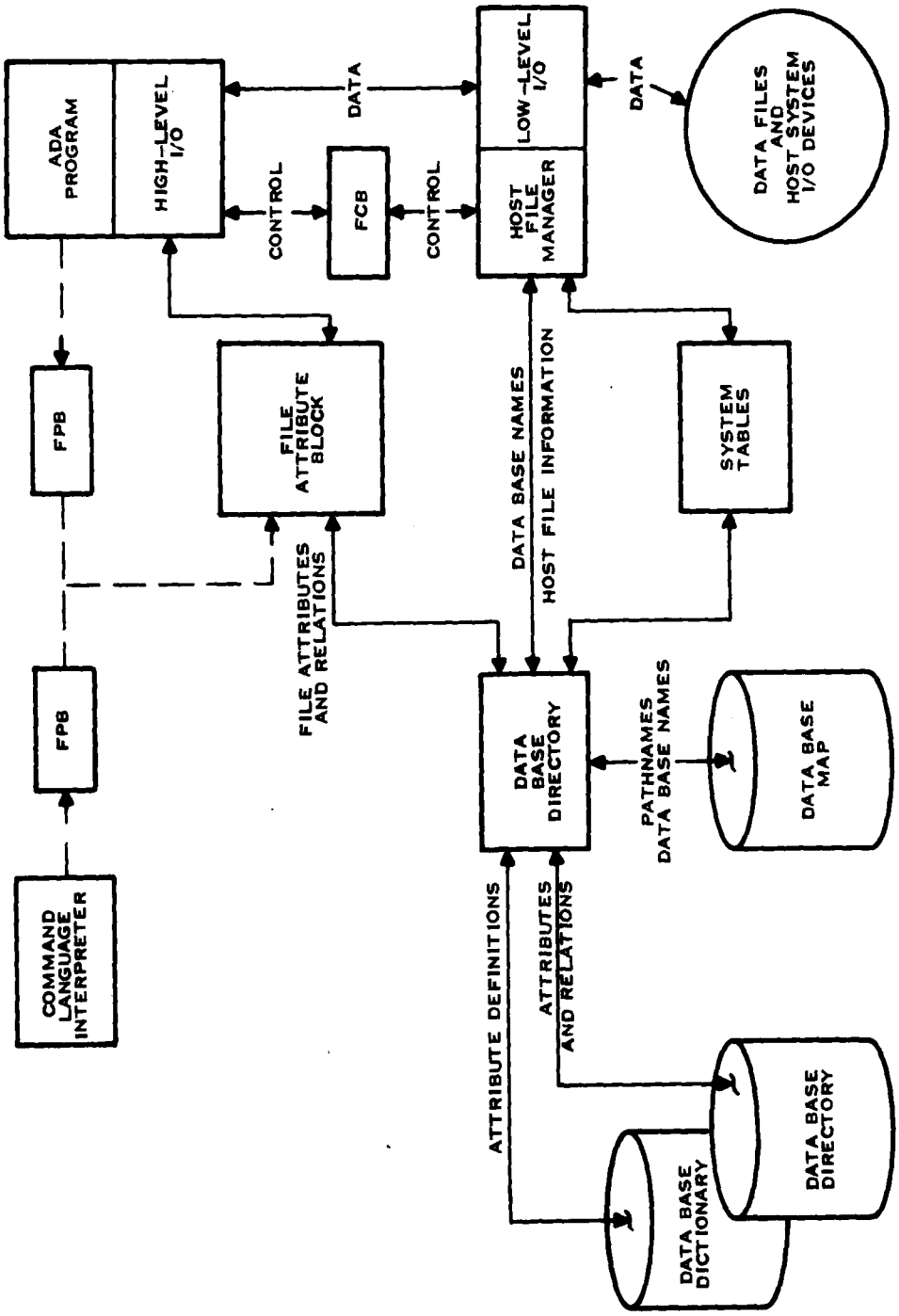


Figure 3-1 Database Directory Interfaces

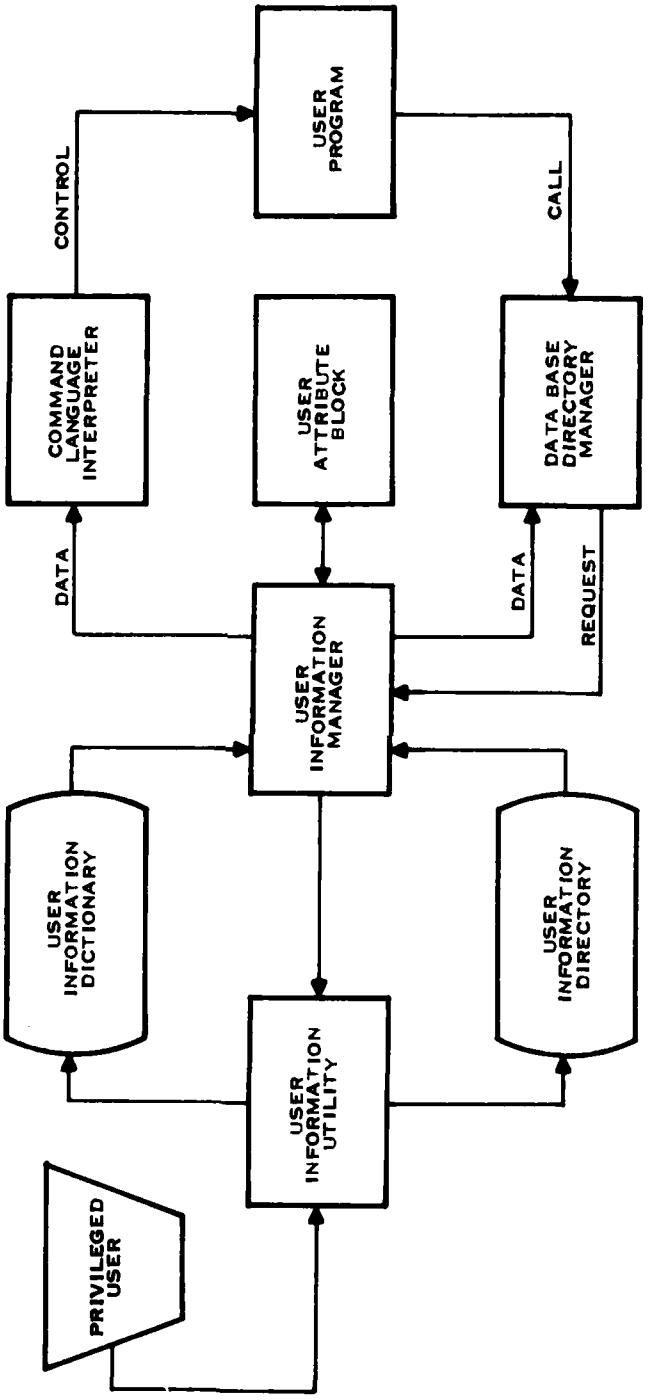


Figure 3-2 User Information Interfaces

3.1.4 Program Interfaces

The Ada Database Subsystem interfaces with the Command Language Interpreter, the I/O component of KAPSE, and Ada programs. The term "Ada Programs" refers not only to "application" or "user" programs but also to the Compiler, Debugger, Editor, and all other APSE tools.

3.1.4.1 Interface with Programs

All Ada programs interface with the database through File Parameter Blocks and File Attribute Blocks as follows:

The File Parameter Block is a data structure created by the Ada compiler when it processes "file declarations" in an Ada program unit. One FPB is created for each file declared. The binder collects all File Parameter Blocks to create a single FPB list so that all files associated with a program are known.

A File Attribute Block is constructed at run time by merging information from already existing file attributes stored in the database and from the information contained in the File Attribute Blocks for the file constructed by the Ada compiler, program binder, or command language interpreter.

The order of merging FPBs to get a FAB is first from compiler-generated FPBs, then from CLI-generated FPBs, and finally compared with information from the directory. Essentially, what is requested is compared with what exists. When a file is opened FAB data is compared with information from the directory. If attribute comparisons are required, exceptions may be raised during this process.

A File Attribute Block contains a fixed area and a flexible area. A File Parameter Block contains only a flexible area. Each entry in a File Attribute Block or File Parameter Block is a couple consisting of the attribute name and the attribute value.

The structure of the fixed area of the File Attribute Block is shown in Figure 3-3 below. By "fixed" it is meant that certain attributes always exist, even if they do not have a value. The flexible area begins at the end of the fixed area and is of similar structure; howvever, there is no order among the attributes.

```
ATTRIBUTE NAME

Pathname                -- location in hierarchical directory
DB name                 -- permanent database name
Kind                    -- device type; location of file
Access                  -- access rights
Creator
Owner
Version
Revision
Category                -- type of file
Last_Update_Date        -- date and time of last update
Last_Access_Date        -- date and time of access
Creation_Date           -- date and time of creation
History
Last_Record             -- number of last record in the file
```

Figure 3-3 Structure of File Attribute Block Fixed Area

3.1.4.2 Interface with I/O Component of KAPSE

Interface between the Ada Database Subsystem and the KAPSE I/O component is provided by a package containing a set of routines which treat the database as a virtual device. This package provides routines that recognize DB_names and pathnames and map them to Host_Filesystem_Names. For subsequent operations, the database will forward the request to the host system. (See Figure 3-1.)

The actual host filesystem is not visible to either the database user or the Ada Database Subsystem. This allows the user the same logical access to objects regardless of their actual location. For example, it does not matter if the desired file is on a disk in the current host computer or if it is in another computer system communicating through some network to the current host computer.

3.1.4.3 Interface with Command Language Interpreter

The Command Language Interpreter (CLI) interfaces with the database subsystem in the following ways:

When the run command is issued, the Command Language Interpreter creates a File Parameter Block and associates file information with it. The Command Language Interpreter also creates a File Parameter Block list so that all files associated with a program are known. Again the system is protected should the Command Language Interpreter create a File Parameter Block which is not processed when subsequent processing takes place.

The Database Directory Manager resolves the File Parameter Blocks into the File Attribute Block as above. Information contained in the Command Language Interpreter created File Parameter Block is next merged. (See Figure 3-1.)

3.1.5 Function Descriptions

The Ada Database Subsystem has the following major components: Database Interface, Database Directory Manager, Database Dictionary Utility, Database Utilities, Host Filesystem Interfaces, User Information Manager, User Information Utility, and Database Archive Manager.

The Database_Interface package consists of the routines for general operations on file attributes and relations. This is the interface between all programs and the Ada Database Subsystem. Included are type declarations, functions to handle database names, functions to obtain the values of file attributes and relations, and procedures to assign file attributes and relations.

The Database Directory Manager is a program consisting of routines to read entries from and write entries into the Database Directory and Database Map and to read entries from the Database Dictionary. It File Parameter Blocks created by the compiler and CLI into the appropriate File Attribute Blocks.

The Database Dictionary Utility is a program consisting of routines to define and check attributes and relations, making appropriate entries in the Database Dictionary.

The Database Archive Manager is a collection of management routines and utilities to manage on-line storage resources efficiently, while preserving the integrity, consistency, and availability of all database objects.

The Database Utilities consist of miscellaneous routines for the manipulation of database objects and directories.

The User Information Manager is a program consisting of routines to extract information in the form of User Attributes and relations from the User Information Directory and User Information Dictionary. This information is placed in a User Attribute Block and communicated to the Command Language Interpreter or to the Database Directory Manager in response to a request. These routines are "read only" forms of the routines in the Database Directory Manager and Database Dictionary Utility.

The User Information Utility is a program consisting of routines to define User Attribute Definitions by making entries in the User Information Directory and User Information Dictionary. These routines are "write only" forms of the routines in the Database Directory Manager and Database Dictionary Utility.

The Host Filesystem Interfaces consists of routines to pass data and control between the Database Subsystem, running Ada programs, and the host filesystem. The general routines permit the exchange of data between the

Database Directory Manager and Host Filesystem Interfaces in the form of DB names and host file information, or in the form of system tables. These routines also permit the exchange of data between the objects (data files) in physical storage and the High-Level I/O via the I/O component of KAPSE.

3.2 Detailed Functional Requirements

The Ada Database Subsystem serves as the repository for all user- and system-generated data and as a primary medium of communication between APSE components. In general, the Database Interface is the means of communication between users and tools and the Database Subsystem. The routines of all other components of the Ada Database Subsystem are not accessible by users and tools.

3.2.1 Database Interface - Introduction

The Database Interface is a package consisting of routines for general operations on file attributes and relations. This is the interface between all Ada programs and the Database Subsystem. Included are type declarations, functions to handle database names, functions to obtain the values of file attributes and relations, and procedures to assign file attributes and relations.

Type definitions for the Database Interface are:

```
type DB_NAME   is limited private;
type ATTRIB    is limited private; -- form is character string
type RELATN    is limited private; -- form is character string
type DB_LIST   is array (NATURAL range <>) of DB_NAME;
type RELATIVES is access DB_LIST;
```

The functions to handle database names are:

```
function  IDENT  (FILE: in IN_FILE)      return DB_NAME;
function  IDENT  (FILE: in OUT_FILE)     return DB_NAME;
function  IDENT  (FILE: in INOUT_FILE)   return DB_NAME;
function  IDENT  (NAME: in STRING)       return DB_NAME;
function  NAME   (FILE: in DB_NAME)      return STRING;
```

3.2.1.1 Function ATTRIBUTE

```
function ATTRIBUTE
      (FILE: in DB_NAME;  ATT: in ATTRIB)  return INTEGER;
function ATTRIBUTE
      (FILE: in DB_NAME;  ATT: in ATTRIB)  return BOOLEAN;
function ATTRIBUTE
      (FILE: in DB_NAME;  ATT: in ATTRIB)  return STRING;
```

This routine returns the value of the specified attribute of the object from its File Attribute Block. Attribute values of enumerated types are returned as strings (literal images).

3.2.1.2 Function RELATIONS

```
function RELATIONS
      (FILE: in DB_NAME;  REL: in RELATN)  return RELATIVES;
```

Given the name of a relation, this routine returns a list of database names. The function allows the user to fetch the desired relation(s) or relative(s) and to check for the existence of specified relationships.

3.2.1.3 Procedure SET_ATTRIBUTE

```
procedure SET_ATTRIBUTE
      (FILE: in DB_NAME;  ATT: in ATTRIB; VAL: in INTEGER);
procedure SET_ATTRIBUTE
      (FILE: in DB_NAME;  ATT: in ATTRIB; VAL: in BOOLEAN);
procedure SET_ATTRIBUTE
      (FILE: in DB_NAME;  ATT: in ATTRIB; VAL: in STRING);
```

This routine sets an attribute value for the database object specified by the database name. The attribute must have previously been defined in the dictionary, but may or may not currently exist for the object. The user must have the right to set the value for the attribute.

3.2.1.4 Procedure SELECT_OBJECTS

```
type ATT_LIST is limited private;  -- form is a linked list
                                   -- of records.  Each
                                   -- record contains a
                                   -- triple giving:
                                   --    attribute
                                   --    relational operator
                                   --    value

procedure SELECT_OBJECTS
          (NAMES:   in     STRING;
           PARAMS:  in     ATT_LIST;
           OBJECTS: in out DB_LIST);
```

This procedure is an inquiry of the database and defines a partition based on attributes and their corresponding values. Starting with a pathname and given a list of specifications as a set of attribute-value criteria, the procedure returns a list of all DB_names satisfying the request. The list may name zero, one, or many database objects. If the starting node (pathname) is omitted, the contents of the entire directory will be searched as a default.

3.2.1.5 Procedure MAKE_RELATION

```
procedure MAKE_RELATION
      (FROM: in DB_NAME; REL: in STRING; TO: in DB_NAME);
```

This routine relates two objects under the named relation. The relation must have been previously defined in the dictionary.

3.2.1.6 Procedure BREAK_RELATION

```
procedure BREAK_RELATION
      (FROM: in DB_NAME; REL: in STRING; TO: in DB_NAME);
```

This routine eliminates the named relationship between two objects. The relation name must be defined in the dictionary.

3.2.2 Database Directory Manager - Introduction

A directory is a special object in the Ada database. It is itself a form of relational database whose entries contain information pertaining to Ada database objects. There is one and only one directory entry for each oject in the Ada database. A directory entry contains the attributes and

relations which describe a database object.

The Database Directory Manager consists of routines to read from and write to the directory and to read the database dictionary. Included are routines to access an entry, create or delete an entry, obtain the values of an object's attributes and relations as described in the entry, and modify the values of attributes and relations. These routines are not directly accessible to users.

3.2.2.1 Procedure FIND_ENTRY

```
procedure FIND_ENTRY
        (NAME: in  STRING; FAB: out FILE_ATTRIBUTE_BLOCK);
```

This routine traverses the directory until a specified object is found. Once found, the Directory Manager reads the directory entry and constructs a File Attribute Block (FAB) containing the information to enable access of that object's attributes and relations.

2.2.2 Procedure ADD_ENTRY

```
procedure ADD_ENTRY
        (NAME: in STRING; FAB: out FILE_ATTRIBUTE_BLOCK);
```

This routine inserts the specified entry into the directory. The pathname and other system-defined, system-maintained attributes are automatically added, and a complete FAB is returned.

3.2.2.3 Procedure DELETE_ENTRY

```
procedure DELETE_ENTRY(NAME: in STRING);
```

This routine deletes the named entry from the directory, provided it has no relations. Otherwise, the entry is marked for deletion. This prevents access by any other program. After all relations are broken, the entry may be removed from the directory.

3.2.2.4 Procedure UPDATE_HISTORY

```
type HISTORY_RECORD (INPUTS: NATURAL) is
   record
      PROGRAM_NAME:  DB_NAME;
      INPUT-NAMES:   DB_LIST (1..INPUTS);
      UPDATE_USER:   USER_ID;
      UPDATE_DATE:   TIME_STAMP;
      REVISION:      INTEGER;
   end record;

procedure UPDATE_HISTORY
            (FILE: in DB_NAME;
             HIST: in HISTORY_RECORD);
```

This procedure updates the history record of a database object. This routine can determine the names of programs, determine names of all input files, determine USER_ID, insert version/revision codes, and construct/insert date and time stamps.

3.2.3 Database Dictionary Utility - Introduction

The dictionary is a special object in the Ada database. It contains the definitions of all database attributes and relations. A dictionary entry for an attribute will include its name, data type, and other appropriate information equivalent to that found in a symbol table. ATTRIBUTE_DEFINITION is a record containing the global type definitions for the Database Dictionary Utility procedures.

```
type ATTRIBUTE_DEFINITION is
   record
      ATTRIBUTE_TYPE:  (STRING, INTEGER, BOOLEAN, RELATION);
      ATTRIBUTE_ACCESS:     (PUBLIC, RESTRICTED, EXCLUSIVE);
      ATTRIBUTE_DEFINER:    (SYSTEM, USER);
      ATTRIBUTE_MAINTAINER: (SYSTEM, USER);
      TEST_REQUIRED:        BOOLEAN:= FALSE;
   end record;
```

3.2.3.1 Procedure DEFINE_ATTRIBUTE

```
procedure DEFINE_ATTRIBUTE
       (ATTRIBUTE_NAME:   in  STRING;
        ATTRIBUTE_ENTRY:  in  ATTRIBUTE_DEFINITION;
        ATTRIBUTE_INDEX:  out INTEGER);
```

This routine makes an entry in the Dictionary indicating the name and type of a new attribute. A relation is a special attribute type.

3.2.3.2 Procedure CHECK_ATTRIBUTE

```
procedure CHECK_ATTRIBUTE
      (ATTRIBUTE_NAME:   in     STRING;
       ATTRIBUTE_ENTRY:  in out ATTRIBUTE_DEFINITION;
       ATTRIBUTE_INDEX:  out    INTEGER);
```

This routine is used to obtain the definition of an attribute. A "symbol table lookup" is performed for ATTRIBUTE_NAME.

3.2.4 Database Archive Manager - Introduction

The Database Archive Manager (DAM) is a collection of management routines and utilities to manage on-line storage resources efficiently, while preserving the integrity, consistency, and availability of all database objects.

For text objects, such as the source text of an Ada program module, successive revisions may be represented by incremental changes to a baseline database object. This technique is useful when each line of text is uniquely identified and tracking of changes to individual lines of text is required, as in the case in MIL-STD-1679. Each set of changes is stored in a single archive file from which any particular revision level may readily be extracted. These archiving techniques have the advantage of retaining a large number of revisions in minimal on-line storage, at the expense of some processing overhead.

Objects that are not suited to incremental change archiving must maintain multiple revisions as separate but related objects in the database. A general-purpose utility for copying database objects between mass storage devices and between mass storage and backup media is provided. The utility has the ability to copy single objects or various other partitions of the database along with appropriate histories, directories, and attribute records.

3.2.4.1 Package ARCHIVE_MANAGER

The ARCHIVE_MANAGER package provides utilities to initialize and update an archive file and to extract selected versions (revisions) of the stored text from the file. (See Figure 3-4.)

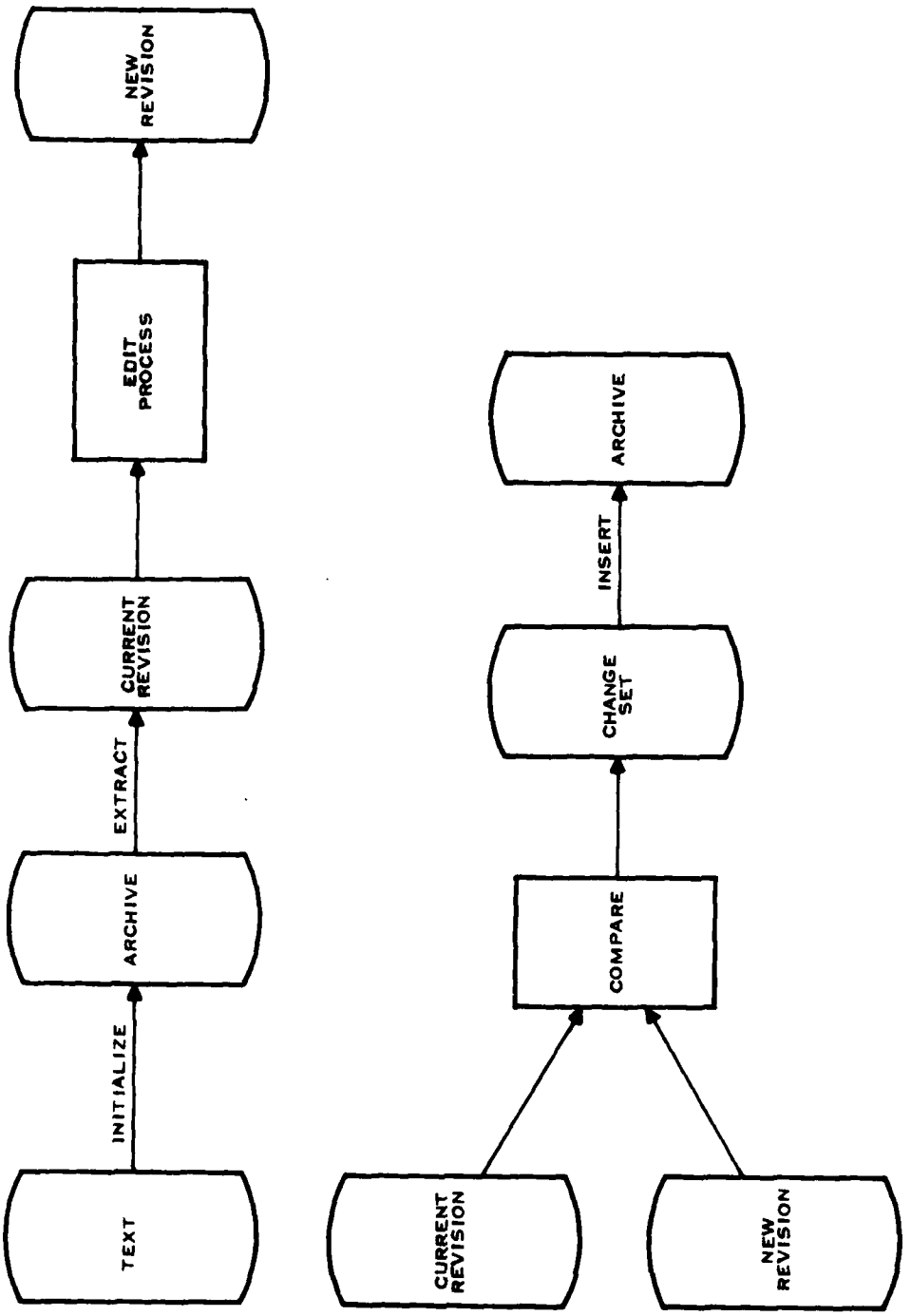


Figure 3-4 Archive File Process

```
package ARCHIVE_MANAGER is

procedure INITIALIZE --  initializes a new archive file
          (ARCHIVE_FILE:   in STRING;
           STARTING_FILE:  in STRING;
           REVISION_LEVEL: in INTEGER:= 1);

procedure UPDATE     --  puts change sets into
                     --  the archive file
          (ARCHIVE_FILE: in STRING;
           UPDATE_FILE:  in STRING;
           NEW_REVISION: in INTEGER);

procedure EXTRACT    --  gets the requested revision
                     --  from an archive file
          (ARCHIVE_FILE:     in STRING;
           DESTINATION_FILE: in STRING;
           REVISION_LEVEL:   in INTEGER);

end ARCHIVE_MANAGER;
```

3.2.4.1.1 Structure of the Archive File

The structure of the archive file differs from the structure of a baseline textual database object. Each archive file record consists of two fields: a text field and a control field. The text field is the same as a record in a textual database object. It contains the "contents" of the object. The control field is a vector indicating the range of the revision levels in which the specified record appears. The full structure is shown in Figure 3-5.

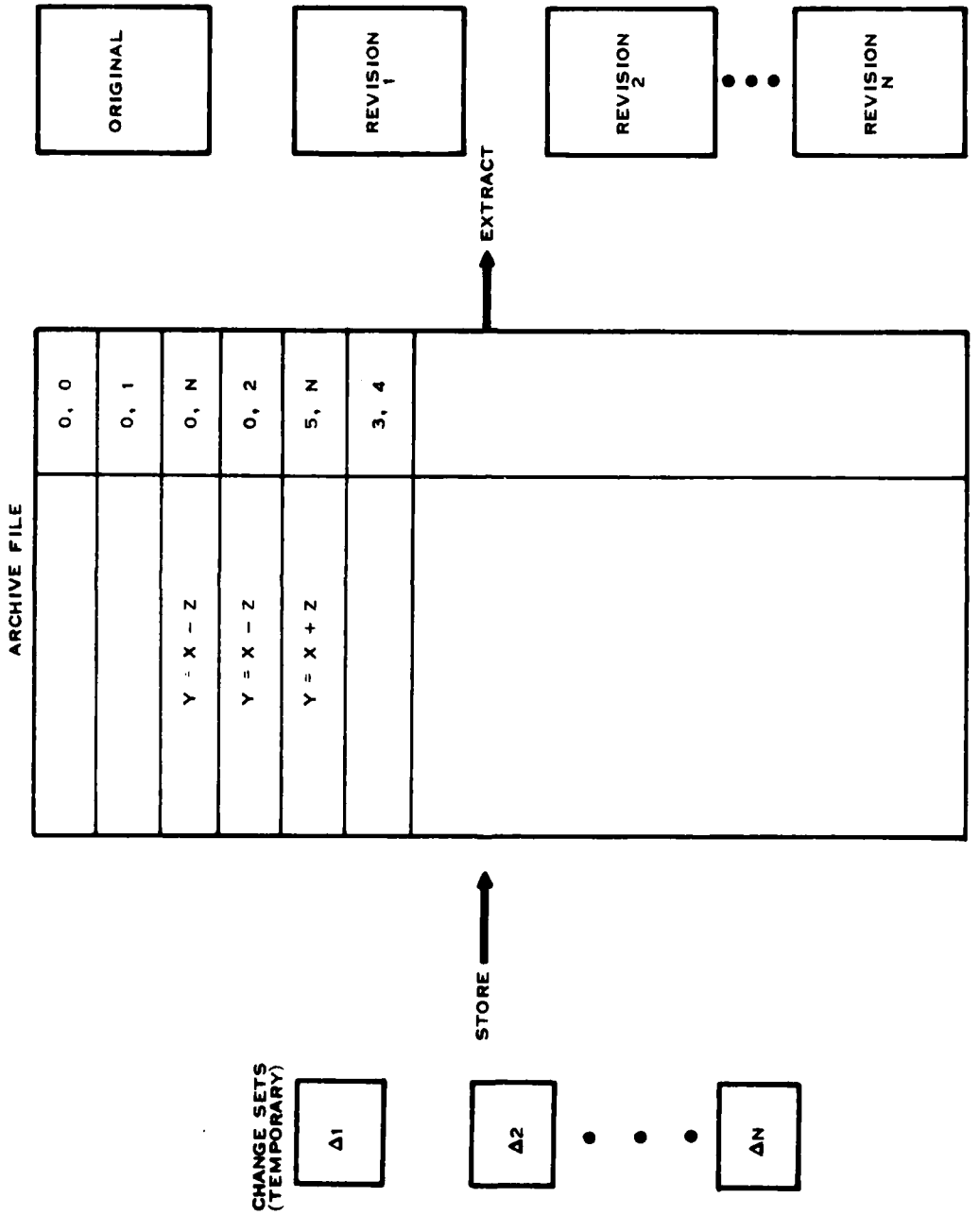


Figure 3-5 Structure of the Archive File

3.2.4.1.2 Procedure INITIALIZE

The INITIALIZE procedure is used to create a new archive file. This procedure sets up the archive file structure and copies the text file specified by the user into the archive file. Archive data is also inserted, including the date and time that the archive was created, the identification data of the user creating the archive, and any user specified attributes.

3.2.4.1.3 Procedure UPDATE

The UPDATE procedure just inserts the change set in the archive. COMPARE builds the change file.

3.2.4.1.4 Procedure EXTRACT

This procedure extracts the text file designated by a specific revision revision level from the archive file. The information stored in the control field of the archive file (see Figure 3-5) is used to determine which records belong to the requested revision; these records are copied to the output file. Only the text fields of the selected records are copied.

3.2.5 Database Utilities - Introduction

The Database Utilities are a collection of routines that allow users to view (list) or copy the contents of database directories and objects, and to verify the copies made. This section describes these general-purpose routines:

* SHOW -- display the contents of a directory or object
* COPY -- copies contents from one object to another
* COMPARE -- compares the contents of two database objects
* BACKUP -- special form of COPY
* RESTORE -- special form of COPY
* VERIFY -- special form of COMPARE

3.2.5.1 Procedure SHOW

```
procedure SHOW(OBJECT_PATHNAME: in STRING; OUTPUT: in STRING);
```

This procedure displays the contents of a database object. The object may be any database object including a directory. OUTPUT names the file or device on which the contents will be displayed. If the object is not a textfile, then the hexadecimal representation of the memory or image map will be displayed.

3.2.5.2 Procedure COPY

```
procedure COPY(FROM: in STRING; TO: in STRING);
```

The COPY procedure is used to copy database objects to and from various storage media and devices. The source and destination must be explicitly stated. If the destination is a database object pathname, COPY creates a new database object and attempts to give it the designated name. If an object by this name already exists, an exception may be raised.

3.2.5.3 Procedure COMPARE

```
procedure COMPARE                    -- compares two objects
      (OLD_FILE:     in STRING;      -- source object
       NEW_FILE:     in STRING;      -- source object
       UPDATE_FILE:  in STRING);     -- internally generated
```

The COMPARE procedure performs a comparison between two files and generates an update file for an archive. A listing file is also produced.

* The LISTING file shows records that were inserted, deleted, or not changed. A symbol will be added to the right side of each record to indicate the occurence of an insertion (*I*) or deletion (*D*).

* The UPDATE file will specify only which lines were inserted or deleted. The information contained in those lines that were inserted will also appear in this file.

3.2.5.4 Procedure BACKUP

```
procedure BACKUP(LIBRARY: in STRING; MEDIA: in STRING);
```

The BACKUP procedure is a specialized call for the COPY procedure. The formal parameters are such that the object(s) in LIBRARY are copied to the object specified in MEDIA. Typically, MEDIA contains the pathname of an object that represents either sequential storage media or removable disk pack.

3.2.5.5 Procedure RESTORE

```
procedure RESTORE(LIBRARY: in STRING; MEDIA: in STRING);
```

The RESTORE procedure is a specialized call for the COPY procedure. The formal parameters are such that the object in MEDIA is copied to the object(s) specified in LIBRARY. Typically, MEDIA contains the pathname of

an object that represents either sequential storage media or removable disk pack.

3.2.5.6 Procedure VERIFY

```
procedure VERIFY                    -- calls procedure COMPARE
     (COMMAND:       in STRING;     -- values are BACKUP or COPY
      FIRST_OBJECT:  in STRING;     -- SEQUENTIAL or MASTER
      SECOND_OBJECT: in STRING);    -- DIRECTORY or COPY
```

This procedure calls the procedure COMPARE and locally generates the two temporary internal objects needed by COMPARE - LISTING file and UPDATE file. For BACKUP, the FIRST_OBJECT is the pathname for the sequential access device on which the "backup" copy of the database object in SECOND_OBJECT is being performed. For COPY, the FIRST_OBJECT is the pathname of the database object serving as a "master" from which the database object in SECOND_OBJECT is copied.

3.2.6 User Information Manager - Introduction

The User Information Directory is a special kind of object in the Ada database. It is a form of relational database whose objects contain information pertaining to users. There is one and only one user directory entry for each user in the Ada database. A user directory entry contains the attributes and relations which describe a user (User Attributes and Relations).

The User Information Dictionary is also a special kind of object in the Ada database. It contains the definitions of user attributes and relations specified for a particular set of users. Each user information directory is linked to a specific user information dictionary. A user dictionary entry for a user attribute will include its name, data type, and other appropriate information equivalent to that found in a symbol table.

The User Information Manager consists of routines to read from the user information directory and user information dictionary. Included are routines to access an entry and interrogate a user's attributes and relations. Type definitions for these routines are:

```
type USER_ATTRIBUTE_BLOCK is limited private;
type USER_NAME            is limited private;
type ATTRIB               is limited private;
type DB_LIST              is limited private;
type USER_ATTRIBUTE_DEFINITION is
  record
    USER_ATTRIBUTE_TYPE: (STRING, INTEGER, BOOLEAN, RELATION);
    USER_ATTRIBUTE_ACCESS: (PUBLIC, RESTRICTED, EXCLUSIVE);
  end record;
```

3.2.6.1 Function USER_ATTRIBUTE

```
function USER_ATTRIBUTE
      (FILE: in USER_NAME;  ATT: in ATTRIB)  return INTEGER;
function USER_ATTRIBUTE
      (FILE: in USER_NAME;  ATT: in ATTRIB)  return BOOLEAN;
function USER_ATTRIBUTE
      (FILE: in USER_NAME;  ATT: in ATTRIB)  return STRING;
```

This routine returns the value of the specified user attribute of the object indicated. The value is returned in string form along with its type. System attribute values are not returned to non-privileged users.

3.2.6.2 Function USER_RELATIONS

```
function USER_RELATIONS
    (FILE: in USER_NAME;  REL: in DB_LIST)  return RELATIVES;
```

Given the name of a user relation, this routine returns a list of relatives. The function allows the user to fetch the desired user relation(s) or relative(s) and to check for the existence of specified relationships. System relations are skipped for non-privileged users.

3.2.6.3 Procedure CHECK_USER_ATTRIBUTE

```
procedure CHECK_USER_ATTRIBUTE
    (USER_ID:                in     STRING;
     USER_ATTRIBUTE_NAME:    in     STRING;
     USER_ATTRIBUTE_ENTRY:   in out USER_ATTRIBUTE_DEFINITION);
```

This routine is used to verify that a type definition of an attribute agrees with the one with which it was previously defined.

3.2.6.4 Procedure FIND_USER_ENTRY

```
procedure FIND_USER_ENTRY
      (NAME: in  STRING; UAB: out USER_ATTRIBUTE_BLOCK);
```

This routine traverses the user directory until a specified object is found. Once found, the User Information Manager reads the user information directory entry and enters into a User Attribute Block (UAB) the information to enable access of that user's attributes and relations.

3.2.7 User Information Utility - Introduction

The User Information Directory and User Information Dictionary are special kinds of objects in the Ada database. They have been described previously.

The User Information Utility consists of routines to write to the user information directory and user information dictionary. Included are routines to create or delete a user, interrogate a user's attributes and relations, and modify a user's attributes and relations. Type definitions for these routines are:

```
type ATTRIB                is limited private;
type USER_ATTRIBUTE_BLOCK is limited private;
type USER_NAME             is limited private;
type USER-ATTRIBUTE_DEFINITION is
  record
    USER_ATTRIBUTE_TYPE: (INTEGER, BOOLEAN, STRING, RELATION);
    USER_ATTRIBUTE_ACCESS: (PUBLIC, RESTRICTED, EXCLUSIVE);
  end record;
```

There are no DELETE_USER_ATTRIBUTE_DEFINITION functions! User Attribute and Relation Definitions cannot be deleted or changed.

3.2.7.1 Procedure ADD_USER

```
procedure ADD_USER
    (USER_PATHNAME: in STRING; UAB: out USER_ATTRIBUTE_BLOCK);
```

This routine inserts the specified user into the user directory. The pathname attribute is automatically added, and a complete UAB is returned.

3.2.7.2 Procedure DELETE_USER

```
procedure DELETE_USER(USER_PATHNAME: in STRING);
```

This routine deletes the named user from the user directory, provided it has no relations. Otherwise, the user entry is marked for deletion. This prevents user access. After all relations are broken, the user entry may be removed from the user information directory.

3.2.7.3 Procedure DEFINE_USER_ATTRIBUTE

```
procedure DEFINE_USER_ATTRIBUTE
    (USER_ATTRIBUTE_NAME:   in     STRING;
     USER_ATTRIBUTE_ENTRY:  in out USER_ATTRIBUTE_DEFINITION;
     USER_ATTRIBUTE_INDEX:  out    INTEGER);
```

This routine makes an entry in the User Information Dictionary indicating the name and type of new attribute.

3.2.7.4 Procedure SET_USER_ATTRIBUTE

```
procedure SET_USER_ATTRIBUTE
      (FILE: in USER_NAME;  ATT: in ATTRIB; VAL: in STRING);
procedure SET_USER_ATTRIBUTE
      (FILE: in USER_NAME;  ATT: in ATTRIB; VAL: in BOOLEAN);
procedure SET_USER_ATTRIBUTE
      (FILE: in USER_NAME;  ATT: in ATTRIB; VAL: in INTEGER);
```

This routine adds an attribute to the object specified by the input. The attribute must have been previously defined in the dictionary, but need not currently exist for the specified user. If the attribute/relation list does not exist, one will be created. Only privileged users can add system or read only attributes.

This routine also deletes a specific attribute from the directory entry spcified by the input. System attributes can only be deleted by privileged users. "Pathname" cannot be deleted.

3.2.7.5 Procedure MAKE_USER_RELATION

```
procedure MAKE_USER_RELATION
      (FROM: in USER_NAME; REL: in STRING; TO: in USER_NAME);
```

This routine relates two users under the named relation. The relation must have been previously defined in the user information dictionary.

3.2.7.6 Procedure BREAK_USER_RELATION

```
procedure BREAK_USER_RELATION
        (FROM: in USER_NAME; REL: in STRING; TO: in USER_NAME);
```

This routine eliminates the named relationship between two users. The user *relation* must have been previously defined in the user information dictionary.

3.2.8 Host Filesystem Interfaces - Introduction

The Host Filesystem Interfaces consist of routines to pass data and control between the Ada Database Subsystem, running Ada programs, and the host filesystem. The general routines permit the exchange of data between the Database Directory Manager and Host Filesystem Interfaces in the form of DB_names and host filesystem information, or in the form of system tables. These routines also permit the exchange of information between the objects (data files) in physical storage and the High-Level I/O via the Low-Level I/O.

3.2.8.1 High-Level I/O Interface

Input-output facilities are predefined in the Ada language by means of two packages. The generic package INPUT_OUTPUT defines a set of input-output primitives applicable to objects whose contents are of a single type. Additional primitives for text input-output are supplied in the package TEXT_IO. These facilities are described in Chapter 14 of the Reference Manual for the Ada Programming Language [DoD80B].

3.2.8.2 Low-Level I/O Interface

An operation acting on a physical device is handled by using one of the predefined procedures SEND_CONTROL and RECEIVE_CONTROL. These are both overloaded procedures. Procedure SEND_CONTROL is used to send control information to a physical device. Procedure RECEIVE_CONTROL is used to monitor the execution of an input-output operation by requesting information from the physical device. These procedures are in the standard package LOW_LEVEL-IO and have two parameters identifying the device and the data.

The visible part of the package defining these procedures is:

```
package LOW_LEVEL_IO is
   -- declarations of the possible types for DEVICE and DATA;
   -- declarations of overloaded procedures for these types;
   procedure SEND_CONTROL    (DEVICE: in     device_type;
                              DATA:   in out data_type);
   procedure RECEIVE_CONTROL (DEVICE: in     device_type;
                              DATA:   in out data_type);
end LOW_LEVEL_IO;
```

The bodies of the procedures SEND_CONTROL and RECEIVE_CONTROL are specified in the body of the package LOW_LEVEL_IO.

3.2.8.3 System Tables

APSE Executive routines are provided to manage the processing and flow of Device I/O control, file utility control, and File I/O control. These executive routines require searches of the appropriate system tables: the Logical Device Table and the Physical Device Table. A Physical Device Table is a data structure that represents a physical device to the Executive. In addition to containing information describing the device, the Physical Device Table is used as a workspace by the executive device service routine. A Logical Device Table is built in the system table area to represent the logical unit to the Ada Integrated Environment.

Logical Device Tables, Physical Device Tables, and APSE Executive routines are discussed in detail elsewhere.

SECTION 4

QUALITY ASSURANCE PROVISIONS

4.1 Introduction

Testing of the Ada Database Subsystem shall be in accordance with the schedule, procedures and methods set forth in the following documents:

1. Contractor's Computer Program Development Plan (CPDP);
2. Computer Program Test Plan for this CPCI;
3. Computer Program Test Procedures for this CPCI.

Testing of the Ada Database Subsystem shall be performed at three levels:

1. Computer program component test and evaluation;
2. Integration test, involving all components of the CPCI;
3. Computer program acceptance testing, involving the APSE.

4.1.1 Computer Program Component Test and Evaluation

This level of testing supports development. Each component of the Database Subsystem shall be tested as a stand-alone program before integration. This testing shall concentrate on areas where new algorithms have been developed or where there is relatively high risk. Examples of such areas are:

* Directory Manager performance
* Multi-user database access/update
* Access control

The test bed for unit testing shall be the parts of the Database Subsystem that are complete at test time, together with special purpose drivers required for each test.

Test results shall be recorded in informal documentation; formal test reports are not required.

4.1.2 Integration Testing

This level of testing supports integration and prepares for acceptance tests. During this testing, Ada Database Subsystem components shall be integrated, one at a time, and run with previously tested subsets of the complete Ada Database Subsystem. Testing at this level shall follow the test plan and procedures for the CPCI. Formal test reports are not required.

4.1.3 Formal Acceptance Testing

This testing assures that the Database Subsystem conforms to the requirements in the Type A and B5 specifications. A formal test plan and test procedures shall be generated and used to insure conformance to the requirements. Acceptance tests shall be defined to incrementally test major functional components of the Database Subsystem. Acceptance testing shall be documented in accordance with the Computer Program Development Plan and Computer Program Test Plans, and delivered to the Government with final system documentation.

4.2 Test Requirements

Unit testing and integration testing shall be performed using the developed Database Subsystem and needed drivers. While testing shall not use formal test plans, testing shall keep the final acceptance tests in mind. Unit tests and integration tests consist of three primary parts:

* Function tests
* Utility tests
* Special purpose tests designed to exercise Database Subsystem features not otherwise adequately covered.

4.2.1 Function Tests

Tests of each Database Subsystem function include testing each routine in the Database Interface package, the Directory Manager, the User Information Manager, the Database Archive Manager, and the Host Filesystem Interfaces.

4.2.2 Utility Tests

Tests of database/user utility programs include testing each routine in the Dictionary Utility, the User Information Utility, and the Database Utilities.

4.2.3 Rehosting Tests

Parallel sets of function tests, utility tests, and special tests shall be run on the IBM 370 and the Interdata 8/32 Database Subsystems. Components of the 370 version may be used to simulate or provide drivers for components not yet rehosted on the Interdata 8/32, during unit testing.

4.3 Acceptance Test Requirements

The acceptance tests shall be run according to the contractually developed test plan. The acceptance test shall consist of the three groups of tests described above: function tests, utility tests, and special purpose tests designed to test particular features.

4.3.1 Performance Requirements

The performance of the Ada Database Subsystem shall be measured in terms of its use to host system resources and in efficiency of the software service it provides.

The Government shall specify the machine and operating system configurations for the initial Ada Integrated Environment host systems. Acceptance test plans shall specify Database Subsystem performance requirements in terms of processing speed and memory use in these host systems.

4.4 Independent Validation and Verification

An independent validation and verification (IV&V) contractor, if one participates in the Ada Integrated Environment program, may perform independent testing of the Ada Database Subsystem using any of the tests descibed above or additional procedures.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.


FILMED
8